SIMULATION STUDIES OF A VIRTUAL MEMORY, TIME-SHARED, DEMAND PAGING OPERATING SYSTEM

J. Winograd, S. J. Morganstein*, R. Herman**
RCA Corporation, Computer Systems Division
Cinnaminson, New Jersey

Summary

SIM/61 is a large (3500 lines of Simscript code), highly detailed simulation model of a virtual memory, time-shared, demand paging operating system. SIM/61 provides the capability for parameterized modeling of both hardware and software. The current model contains algorithms for interrupt analysis, task scheduling, I/O scheduling and demand paging.

This paper reports the results of studies made using SIM/61. The studies fall into two main categories: (1) load and configuration studies and (2) alternate algorithm studies. The approach taken for the former was to establish a fixed load, and measure its performance on various hardware configurations. The results are particularly interesting with respect to the paging capability of various paging device configurations, and various sizes of main memory.

The alternate algorithm study was concerned with task scheduling. In particular, it was shown that a minor change to the original task scheduling algorithm provided a great deal of flexibility in enabling system resource utilization to be biased toward either batch or interactive processing, and in varying degrees.

Introduction

This paper reports the results of simulation experiments which were carried out using SIM/61, a highly detailed simulation model of a virtual memory, time-shared, demand paging operating system. A brief description of SIM/61 is provided, while the reader is referred to (1) for a more detailed discussion of the model. The operating system simulated is also briefly discussed.

SIM/61: The Operating System

The operating system modeled is a virtual memory, demand paging system, which concurrently supports batch and interactive processing. Definitions and algorithms relevant to this paper are described below.

*Present affiliation: International Telephone and Telegraph Company Paris, France
**Present affiliation: Control Data Corporation Toronto, Ontario

Task Scheduler

A task is simply any job in the system, e.g. a batch job spooled in from the card reader or an interactive user at a TTY. The three basic task types are interactive, batch and communications. It is the responsibility of the task scheduler to control the flow of tasks through the system.

The task scheduler maintains several queues, for tasks in various states, with the queues relevant to this discussion being depicted in Figure 1. Note that there are two distinct ready queues, one for batch tasks and one for interactive tasks. Various queue transitions are depicted by the arrows, with a transition into the ready queues implying that a batch task is placed in the batch ready queue, while an interactive task is put in the interactive ready queue. Communications tasks are basically treated like interactive tasks, with the former having higher priority, i.e. communications tasks are linked ahead of interactive tasks on the interactive ready queue.

Crucial to the task scheduler is the "activate decision", performed by a routine known as task activator. Task activator is guided in its decision making by the Working Set Principle[2], i.e. a task will be activated only if its working set size estimator is less than or equal to the amount of unscheduled core. Thus, the essential ingredient of a task which has been activated is that core has been committed to it (although it will utilize its scheduled core via demand paging). When a task is activated, it is said to be an active task, and it must reside on some active queue; similarly, we have inactive tasks on inactive queues. Note that only the active tasks are allowed to compete for resources (CPU, peripheral I/O, and paging). The set of active tasks is what is commonly known as the multiprogramming mix, with the degree of multiprogramming being the number of active tasks. The system allows a completely variable degree of multiprogramming, where at any point in time the degree is basically determined by core size and task sizes.

Pages

Pages are fixed size, viz. 4096 eight-bit bytes. The system distinguishes between two types of pages. A task page is a private, non-shareable page, residing in the task's virtual memory. An exec page is a public, potentially shareable page, residing in the executive's virtual memory (there is a special kind of exec page known as a shared page). Note that an exec page is not associated with the task which suffered the page fault for it. Thus, when referring to the pages of a task, we mean its task pages.

SIM/61: The Model

SIM/61 is a large, highly detailed simulation model, written in Simscript 1.5. It provides the capability for parameterized modeling of both hardware and software, with particular flexibility in the area of simulating various devices and hardware configurations.

SIM/61 contains the basic operating system algorithms for interrupt analysis, task scheduling, I/O scheduling, and paging. The paging routines have been particularly generalized to allow any number of paging devices, many types of devices, and varying paging-device/channel configurations.

A load is presented to SIM/61 by describing any number of classes of tasks, each class containing a specified number of tasks. In a given class, the set of tasks is described by specifying a list of task characteristics, as follows:

(1) task type: type of task, e.g. batch, interactive, communications.
(2) working set size: number of task pages in the working set.
(3) compute time: for interactive tasks, the amount of time required to service each interaction; for batch tasks, a base time for the other parameters.
(4) think time: total time at the terminal between interactions, including type-in and type-out time (irrelevant for batch tasks).
(5) I/O interval: interval at which to initiate a disc I/O.
(6) exec SVC interval: interval at which to issue a supervisor call (SVC), requesting a function of the pageable executive, with attendant probability of a page fault for an exec page.
(7) shared page reference interval: interval at which to reference a shared page, with attendant probability of a page fault.
(8) virtual memory size: total size (number of pages) of the task's virtual memory, i.e. the amount of backing store it requires.

These parameters serve to define a task, with task paging behavior assumed to obey empirical data gathered by Fine, et. al.[3], as shown in Figure 2 (often called Fine's curve).

Task Paging Behavior

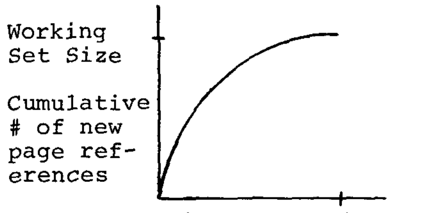


Figure 2

All parameters, except (1) and (8), are drawn from a normal distribution (with specified mean and standard deviation) for each interaction.

SIM/61 simulates the steady state situation, i.e. the number of tasks remains constant for a given run. This is tantamount to saying that LOGONs and LOGOFFs are not simulated.

Experiments

Having provided sufficient background, we now proceed to the various simulation experiments.

Backing Store Study

The purpose of this study is to show how system performance, in a paging bound environment, is affected by various main memory sizes and paging rate capacities. The approach taken was to establish a fixed, paging bound load, and run it on several different hardware configurations, altering main memory size and paging device configuration. The hardware elements remaining unchanged were an RCA 7 processor (fixed point add time of 2.25 us) and two RCA 8590 disc storage units (see Table 2) on a single selector channel, dedicated to user (not paging) I/O.

The details of the simulated load are shown in Table 1. The load is a heavy interactive one, with 64 interactive tasks of varying types (BASIC users, file edit users, compute bound executions, I/O bound executions, paging bound executions). Also, there are two communications tasks and two batch tasks. We wish to emphasize that the load chosen places extremely heavy paging demands on the system, unrealistic for typical user environments. This has been done to provide the paging bound environment fundamental to the study.

Two main memory sizes were simulated, viz. 128 pages (1/2MB) and 256 pages (1MB), along with four paging device configurations:

(1) Two RCA 8580 disc storage units, each on a separate selector channel.
(2) One RCA 8567 drum on a selector channel. The 8567 contains a single page per track, with approximately a 3-4 ms "window" between the end of recorded data and the beginning of the track.

(Since the 8567 was the initial paging device simulated, the paging algorithm was designed for it, using the window time to set up and fire the next paging operation, in an attempt to avoid "missed" drum revolutions.)

(3) Two 8567 drums, each on a separate selector channel.

(4) A hypothetical large auxiliary memory (LAM) (bulk core) on a selector channel.

All paging devices are dedicated to paging traffic. The hardware characteristics of the various paging devices are shown in Table 2.

The eight different configurations were simulated with the standard algorithms. Thus, the basic paging routine, with the "window algorithm" designed for the 8567 drum, was also used with the RCA 8580 and the LAM. Obviously, if a paging routine were being designed specifically for the RCA 8580 or LAM, one would not employ a window algorithm, since there is no window. However, we are not particularly concerned with the actual devices involved, but rather with the continuum of paging transfer capability. That is to say, in the graphs presented below, the actual data points can lose their identity, where we intend one to say "given paging rate x, performance level y can be achieved".

The measured performance results are listed in Table 3 (the data represents nine minutes of simulated time, as the run was for ten minutes, with one minute allowed for "settle down"). Also, three graphs are presented in Figures 3-5, treating paging rate as the critical variable, with various performance measures plotted against the paging rate. The reader is cautioned against drawing the conclusion that as the paging rate increases (as a load factor) performance improves. This, of course, is not the case (that the heavier the load, the better the performance). The issue is that we are paging bound, and as we are given the capability for increased paging (via superior paging device configurations), we can decrease the "paging boundedness", thus improving performance. (Note that a single, fixed load was simulated, implying no variability in the load factor.)

In each graph, two curves were drawn: one for the 1/2MB core size configuration and one for the 1MB case. Figure 3 plots the average response time for interactive tasks. Figure 4 plots the number of terminal interactions processed (i.e. the number of responses given) for the interactive tasks. Figure 5 plots "user" (non-overhead) processor utilization.

The reason why the extra core did not significantly improve performance is easily understood. In the 1MB case, there is 2.6 times the schedulable (i.e. non-resident) core as in the 1/2MB case (208 pages versus 80 pages). Indeed, considering the one-drum run, the measured queue statistics indicate that task activator was able to activate 2.6 times as many tasks (the average number of active tasks was 8.4 versus 3.2). However, since both runs were paging bound, the larger number of active tasks in the 1MB case caused a larger average delay to occur when satisfying a page fault. Again, this is borne out by the queue statistics. For the two runs, the ratio of the average lengths of the paging queue and the ratio of the average times on the paging queue (i.e. the amount required to satisfy a page fault) were both 2.9 (average length: 7.8 versus 2.7; average time: 225 ms versus 77 ms). Thus, although the larger amount of core allowed a higher degree of multiprogramming, this was largely offset by longer waits on the paging queue.

It should be pointed out that both the 1/2MB and 1MB configurations ran with the same algorithms. That is to say, no special effort was made to take advantage of the extra core. Algorithms specifically tailored to handle larger core configurations would probably have resulted in better performance (for example, rather than simply activating more tasks, the extra core could have been utilized by always keeping a batch task active). However, tailoring software to the hardware configuration was not the purpose of this study, although it could be said that a result of the study is that software tailoring must be done to utilize effectively larger main memory configurations.

It is interesting to note that the data from this simulation study is consistent with Denning's theoretical studies (see section in (4) entitled "Relations Among Processor, Memory, Traverse Time"). In particular, Denning shows in (4) that the relationship between throughput and <u>traverse time</u> (time required to satisfy a page fault, i.e. time spent on the paging queue) is linear (assuming main memory size is held constant). In our study, we observe the predicted linear relationship in Figure 4, as the number of interactions processed is a good measure of throughput (of course, the linear relationship holds only while the system is paging bound; in the LAM runs, the system is I/O bound rather than paging bound). Although the curve presented plots paging rate, the linear relationship also holds for traverse time. For example, in the 1MB two-drum run and 1MB one-drum run, the average times on the paging queue (i.e. traverse times) were 110 ms and 225 ms, while the number of interactions processed were, respectively, 2051 and 1003.

There is another interesting performance result. In all but three of the configurations, the system was so busy attempting to process interactive tasks, that the batch tasks never ran. The only configurations which allowed batch to run were the LAM, 1MB case, where batch tasks received 10.0% of the CPU; LAM, 1/2MB, 1.1% of the CPU; and 2 drums, 1MB, 1.6%. The fact that batch tasks did so much better in the LAM, 1MB run is the reason for the sharp rise in the 1MB curve of Figure 5.

These results point out the need for a

balanced hardware configuration. Increasing the amount of a seemingly critical resource may only marginally affect performance, as some other resource may now become the bottleneck (in this case, increasing core has caused paging to become a more severe bottleneck). This is also illustrated by the fact that the largest improvement from 1/2MB to 1MB was observed in the LAM run (note Figure 5). Clearly, LAM provides the most balance for 1MB of core.

Task Activator "N to 1" Algorithm

The task activator first simulated was designed with the aim of providing good response time for interactive tasks. With this in mind, the following simple guideline was developed: never activate a batch task if there is an interactive task awaiting activation (i.e. the batch ready queue is examined only if the interactive ready queue is empty). Initially, it was felt that this policy would not keep batch tasks from getting adequate service. However, it has turned out that with this algorithm a relatively heavy interactive load can completely lock out batch tasks.

A simple change to the task activator can solve this problem. Namely, define a parameter N, with the meaning that after every N activations of interactive tasks, activate 1 batch task (hence, the so-called N to 1 algorithm). Note that an activation for which there was no competition (i.e. either the interactive or batch ready queue was empty) is not counted against N (or against the 1). This is called a free activation.

The N to 1 algorithm was simulated with a load of 20 interactive tasks and 1 batch task. Of the 20 interactives, 6 were BASIC users with a 15 second think time; 6 BASIC with a 10 second think time; 6 paging bound tasks; and 2 compute bound. Again, a heavier than typical load was simulated, so that a backlog on the interactive ready queue was guaranteed, showing the full effect of N to 1.

The hardware environment consisted of an RCA 3 processor (fixed point add time of 8.88 us), 64 pages of main memory (1/4MB), an RCA 8567 drum, dedicated to paging, and two RCA 8590 discs on a single channel, dedicated to user I/O.

For various runs, with different values of N, Table 4 lists the value of N which was input compared with the value of N measured. This result indicates that the N to 1 algorithm was, indeed, working as specified. The difference between the input N and measured N is due to free activations (see the third column of Table 4). Note that only interactive tasks received free activations. This is because there is only one batch task in the system, and when it is waited for a time-slice runout, it is delayed by the system before entering the ready queue (see Figure 1). During this delay, interactive tasks can get free activations, since the batch ready queue is now empty. On the other hand, however, the interactive load is so heavy that there are always tasks on the interactive ready queue (even in the ∞ to 1 case), preventing the batch task from receiving free activations.

The essential results of the N to 1 study are shown in Table 5 and Figures 6, 7 and 8 (again, ten minute runs, with nine minutes of data). Note that N to 1 with $N=\infty$ is equivalent to the original algorithm. The results are quite pleasing in that the desired capability of biasing system performance toward batch or interactive processing was achieved. For example, in the 5 to 1 run, while degrading response time by 1/3 (12.2 to 16.3), batch performance was improved from no service at all to an elapsed running time of about 7 times its stand-alone time. The table and graphs are self-explanatory. It is interesting to note that the relationship of batch elapsed running time to N is linear(Figure 8).

Conclusions

We have presented the results of two simulation studies using SIM/61, a detailed simulation model of a virtual memory, time-sharing operating system. The first study has yielded some interesting data showing the relationship of paging transfer capability to performance in a paging bound environment. Also shown is the effect of two different main memory sizes on performance. Generally, the results indicate that, without changing the software, increased paging transfer capability has a more significant effect on performance than increased core.

The second study has shown that a minor change to the original task scheduling algorithm permits system performance to be biased toward interactive or batch processing, in varying degrees.

Acknowledgements

The authors would like to acknowledge the continuing support of G. Oppenheimer of RCA, who supplied the original impetus and direction for the backing store study. Deep appreciation goes to M. Fogel who has spent many long hours with us analyzing simulation output. Mr. Fogel has that rare capability of deriving tremendous insight from raw data. Thanks go to E. Gloates of RCA, who is currently directing the SIM/61 effort.

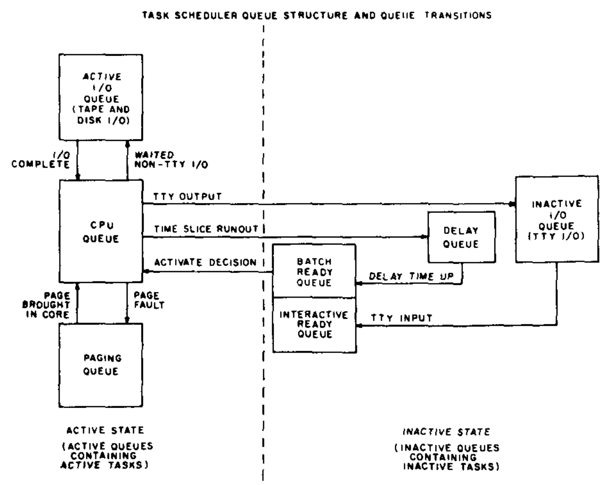
References

(1) Morganstein, S. J., Winograd, J., and Herman, R., SIM/61: A Simulation Measurement Tool for a Time-Shared, Demand Paging Operating System. Proc. ACM Sigops Workshop on System Performance Evaluation (April 1971), 142-172.

(2) Denning, P.J., The Working Set Model for Program Behavior. Comm. ACM 11, 5 (May 1968), 323-333.

(3) Fine, L.H., Jackson, C.W., and McIsaac, P.V., Dynamic Program Behavior Under Paging. Proc. 21st Nat. Conf. ACM, ACM Pub. P-66, 1966, 223-228.

(4) Denning, P.J., Thrashing: Its Causes and Prevention. Proc. AFIPS 1968 Fall Joint Computer Conf., Vol. 33, Part 1, 915-922.


TASK SCHEDULER QUEUE STRUCTURE AND QUEUE TRANSITIONS
ACTIVE I/O QUEUE (TAPE AND DISK I/O)
I/O COMPLETE
WAITED NON-TTY I/O
CPU QUEUE
TTY OUTPUT
TIME SLICE RUNOUT
ACTIVATE DECISION
PAGE BROUGHT IN CORE
PAGE FAULT
PAGING QUEUE
BATCH READY QUEUE
INTERACTIVE READY QUEUE
DELAY QUEUE
DELAY TIME UP
INACTIVE I/O QUEUE (TTY I/O)
TTY INPUT
ACTIVE STATE (ACTIVE QUEUES CONTAINING ACTIVE TASKS)
INACTIVE STATE (INACTIVE QUEUES CONTAINING INACTIVE TASKS)

NOTE 1 THE TERM "TTY" IS MEANT TO ENCOMPASS A BROAD RANGE OF TERMINAL DEVICES.

2 TRANSITIONS TO/FROM THE READY QUEUE IMPLY THAT A BATCH (INTERACTIVE) TASK IS MOVED TO/FROM THE BATCH (INTERACTIVE) READY QUEUE.

Figure 1

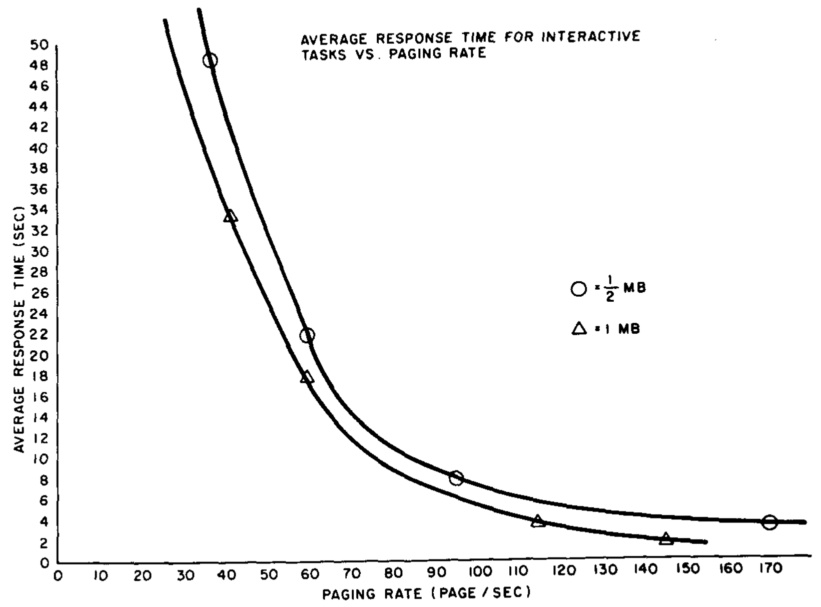


Figure 3

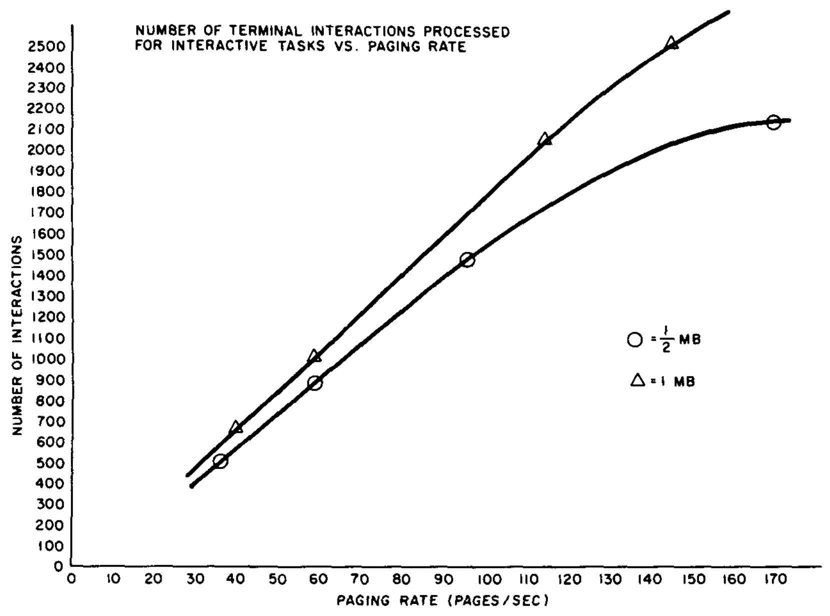


Figure 4

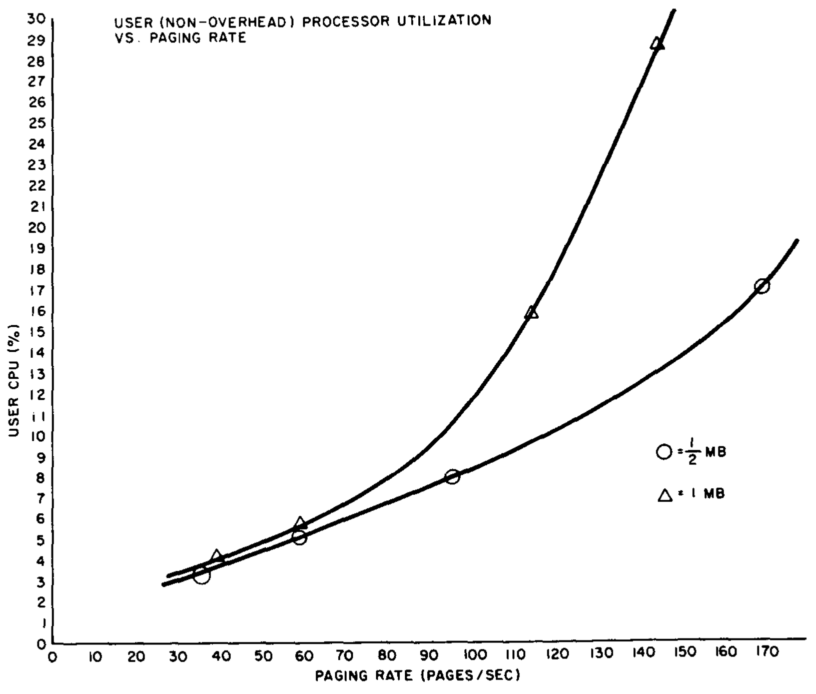


Figure 5

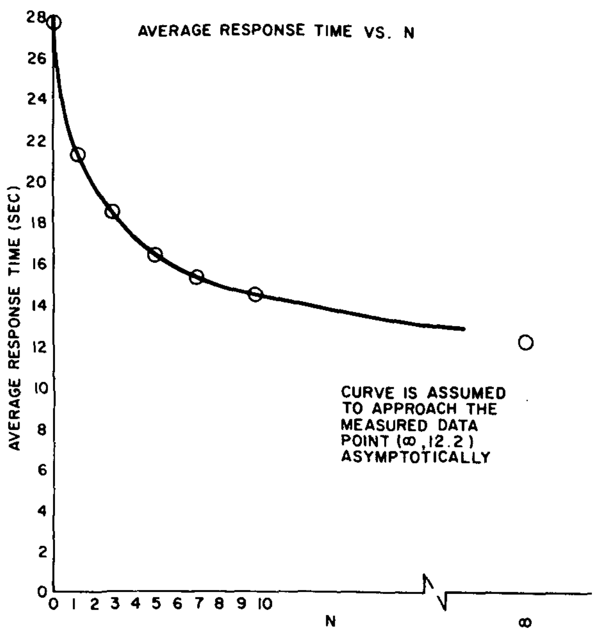


Figure 6

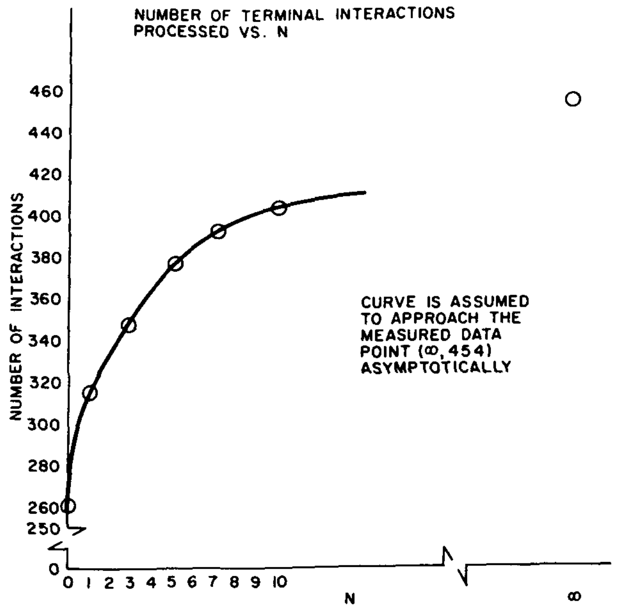


Figure 7

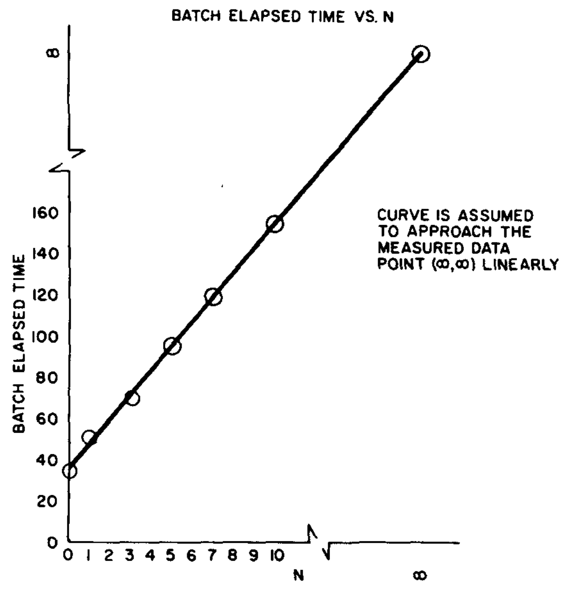


Figure 8

LOAD FOR BACKING STORE STUDY

TASK TYPE*	# OF TASKS	WORKING SET SIZE (PAGES)		COMPUTE TIME (MS)		THINK TIME (SEC)		I/O INTERVAL (MS)		EXEC SVC INTERVAL (MS)		SHARED REF INTERVAL (MS)		TOTAL VIRT MEMORY SIZE (PAGES)
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	
1	9	10	2	25	5	15	3	20.0	5.0	2.0	0.5	3.0	0.7	20
2	9	10	2	20	5	10	2	5.0	1.0	3.0	0.7	10.0	2.5	20
3	5	20	5	300	10	30	5	0	0	5.0	1.0	0	0	40
4	9	15	3	25	5	20	5	2.0	0.5	1.0	0.2	0	0	30
5	9	10	2	10	2	5	2	5.0	1.0	1.5	0.3	0	0	20
6	9	10	2	25	5	25	5	20.0	5.0	2.0	0.5	3.0	0.7	20
7	9	10	2	20	5	20	5	5.0	1.0	3.0	0.7	10.0	2.5	20
8	5	63	10	50	5	30	5	25.0	5.0	5.0	1.0	0	0	126
9	2	20	5	500	50	0	0	10.0	2.5	3.0	0.7	0	0	40
10	2	10	2	10	2	5	2	5.0	1.0	1.5	0.3	0	0	20

PARAMETERS ARE DRAWN FROM A NORMAL DISTRIBUTION WITH MEAN= μ , STANDARD DEVIATION = σ

*1	INTERACTIVE TASK, BASIC USER, SHORT THINK TIME
2	INTERACTIVE TASK FILE EDIT USER, SHORT THINK TIME
3	INTERACTIVE TASK, COMPUTE BOUND EXECUTION
4	INTERACTIVE TASK, I/O BOUND EXECUTION
5	INTERACTIVE TASK, COMMUNICATIONS - LIKE BEHAVIOR
6	INTERACTIVE TASK, BASIC USER, LONG THINK TIME
7	INTERACTIVE TASK, FILE EDIT USER, LONG THINK TIME
8	INTERACTIVE TASK, PAGING BOUND EXECUTION
9	BATCH TASK
10	COMMUNICATIONS TASK

TOTAL OF 64 INTERACTIVE TASKS, 2 BATCH TASKS, 2 COMMUNICATIONS TASKS

Table 1

DEVICE CHARACTERISTICS

	RCA 8567 DRUM	RCA 8580 DISC	RCA 8590 DISC	HYPO-THETICAL IAM
TRANSFER RATE (KB)	333	806	312	900
ROTATION TIME (MS)	17.2	16.7	25.0	---
SEEK TIME - TRACK TO TRACK (MS)	---	10	25	---
SEEK TIME - AVERAGE (MS) *	---	30	75	---
SEEK TIME - MAXIMUM (MS) *	---	55	135	---

* Assumes the entire disc surface is used.

Table 2

PERFORMANCE RESULTS FOR BACKING STORE STUDY

	INTERACTIVE TASK AVG. RESPONSE TIME (SEC)*		PAGING RATE (PAGES/SEC)		USER I/O RATE (2048 BYTE DISC IOs/SEC)		CPU (%)					
							USER		OVERHEAD		IDLE	
PAGING CONFIGURATION	1/2 MB	1 MB	1/2 MB	1 MB	1/2 MB	1 MB	1/2 MB	1 MB	1/2 MB	1 MB	1/2 MB	1 MB
2 RCA 8580 DISCS	48.1 514	33.0 667	36	40	3.4	4.3	3.2	4.0	10.5	11.6	86.3	84.4
1 RCA 8567 DRUM	21.8 883	17.5 1003	58	58	5.4	6.0	5.0	5.6	13.8	14.3	81.2	80.1
2 RCA 8567 DRUMS	7.8 1487	3.9 2051	96	114	8.4	12.0	8.0	15.7	21.4	26.5	70.6	57.8
1 IAM	3.1 2157	1.3 2510	170	145	11.8	20.4	16.8	28.4	41.5	40.7	41.7	30.9

*The figure immediately below average response time is the number of terminal interactions processed (i.e. the number of responses given to interactive tasks) during the measurement period (nine minutes of simulated time).

Table 3

N TO 1 ALGORITHM

VALUE OF N INPUT	VALUE OF N MEASURED	VALUE OF N MEASURED MINUS 1.7
∞	∞	∞
10	11.6	9.9
7	8.7	7.0
5	6.8	5.1
3	4.6	2.9
1	3.0	1.3
0	1.7	0.0

(Note near identity with the first column; 1.7 indicates a virtually constant level of free activations)

Table 4

N TO 1 ALGORITHM

N	∞	10	7	5	3	1	0	BATCH TASK STAND-ALONE
AVERAGE RESPONSE TIME (SEC)	12.2	14.7	15.4	16.3	18.5	21.3	27.8	--
NUMBER OF TERMINAL INTERACTIONS PROCESSED	454	403	392	377	347	316	260	--
BATCH CPU %	0	6.5	8.4	10.4	14.1	19.7	28.2	71.8
BATCH ELAPSED TIME*	∞	154	119	96	71	51	35	14

*BATCH ELAPSED TIME - 1000/BATCH CPU %. Since the steady-state condition is simulated, there is no notion of a batch task running to completion. Thus, "batch elapsed time" is calculated (based on the percentage of CPU time given to the batch task during the run) to provide an elapsed running time for some hypothetical batch job. The various batch elapsed times can be used for comparison of batch throughput.

Table 5